Risk Analysis at Design Level using UML Behavioral Diagrams

Suchitra Kumari Mishra

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

Risk Analysis at Design Level using UML Behavioral Diagrams

Thesis submitted in partial fulfillment
of the requirements for the degree
of

Master of Technology

by

Suchitra Kumari Mishra

Roll no-211CS3296

under the guidance of

Prof. Durga Prasad Mohapatra

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Odisha,769 008, India

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***"Risk Analysis at Design Level using UML Behavioral Diagrams"*** by ***Suchitra Kumari Mishra*** is a record of an original research work carried out by her under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: 30 May 2013

Dr. Durga Prasad Mohapatra
Associate Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

Thank you God for showing me the path...

I owe deep gratitude to the ones who have contributed greatly in completion of this thesis. Foremost, I would like to express my sincere thanks to Prof. Durga Prasad Mohaptra for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I am very much indebted to Prof. Shantanu Kumar Rath, Prof. Ashok Kumar Turuk and Prof. Bibhudatta Sahoo, Prof. Banshidhar Majhi for their encouragement and insightful comments at different stages of thesis that were indeed thought provoking.

I am really thankful to my all friends. My sincere thanks to everyone who has provided me with kind words, a welcome ear, new ideas, useful criticism, or their invaluable time, I am truly indebted.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my family, for their love, patience, and understanding.

Suchitra Kumari Mishra

Abstract

The Risk analysis process recognizes the different type of hazards that can occur and recommend control measures that are frequently used for that hazard. Risk is a measure of the probability and severity of undesired effects. Accomplishment of risk analysis in the early development phases improves resource sharing decisions. This method will aid to find the high-risk components and connectors of the system architecture, so that corrective actions may be implemented to control and improve the development process as well as the quality of the system.

We propose a technique for risk analysis at design level using UML behavioral diagrams. We have used state chart and sequence diagram to find the risk factor of components and connectors involved in the system. Next, we have calculated the risk factor of each scenario of a use case and combined them to obtain the overall risk factor of the targeted system. We have used concurrent control flow graph to evaluate the scenario level risk factor which takes into consideration the concurrent execution of threads. Along with this interaction overview diagram is used to estimate the overall system level risk factor. In our approach, we have also done the sensitivity analysis to find the critical components and connectors with respect to each scenario and also in overall system level. So we can give careful analysis, design, implementation and testing effort to these components and connectors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 1

Introduction

Software testing is a process that detects important bugs with the objective of having better quality software [1]. It can also be defined as a process of investigation conducted to deliver the information about the quality of the product or service under test to the stakeholders [2]. In other words software testing can be defined as the process of validating and verifying a product to check whether it satisfies the requirements that guided its design and development, works as expected, can be implemented with the same features and satisfies the requirements of stakeholders.

Commonly used terms in software testing are error, defect and failure. Software faults may occur whenever a programmer makes an error (mistake) that results in a defect in the source code of software. If the defect is executed in certain situations, the system will produce wrong results causing a failure. Not necessarily all defects will result in failures. For example, defects in dead code will never result in failures because that would not be executed ever. A defect can turn into a failure when the environment is changed.

Software testing can be employed at any time during the development process. Traditionaly the testing process is carried out after the requirements have been defined and the coding process has been completed. Testing after coding is very much expensive and sometimes it is also difficult to test. If we do testing after coding then we have to wait up to coding is completed. During testing if we get any error due to requirement analysis fault then we have to change code, design and requirement which is very much expensive. Sometime code is also not available

such as in (component based software), so testing is not possible. Therefore it is very much useful if testing is done at an early phase of software development. For safety critical systems testing and risk analysis is very much necessary. A safety-critical or life-critical system is a system whose failure may lead to death or severe injury to people or may cause severe loss/damage to equipment or environmentally harm. These systems are designed in such a way that the loss will be less than one life per billion hours of operation. The risk of this sort are managed with the tools and methods of safety engineering.

1.1 Risk Analysis

Software risk management is a part of software project management. It is very important for software projects [3]. Software risk management steps were defined by Barry Boehm [4] and it has two basic steps. The first one is risk assessment and the second is risk control. Risk assessment involves risk identification, risk analysis and risk prioritization. Several techniques can be used for risk identification which produces a list of the project risk items. Risk analysis is the method of discovering risks in applications and ranking them to test. In other words, risk analysis includes the processes concerned with identifying, analyzing and developing security strategy and plans for the factors. A risk is the potential for loss or damage to an organization from materialized threats. Risk Analysis efforts to identify all possible risks and then measure the severity of the risks. A threat is a possible damaging event. When it occurs, it exploits vulnerability in the security of a computer based system. Higher risk value items should be tested early and frequently. Lower risk value items can be tested later. Risks include the factors that might adversely affect project outcomes.

The main objectives of risk analysis are as follows,

- It gives an overview of the general level and pattern of risk faced by the project.
- It helps management to focus on the high-risk items in the list.

- It helps to decide where action is needed immediately, and where action plans should be developed for future activities.
- It facilitates the allocation of resources to support management's decisions.

Risk analysis is useful in many situations, for example:

- During project Planning, it helps to anticipate and neutralize probable problems.
- To decide whether to move ahead with a project or not.
- To improve safety and managing possible risks in the workplace.
- To prepare for happenings such as technology or equipment failure, theft, staff issue, or natural disasters.
- To plan for changes in the environment, such as new competitors entering into the market, or variations to government policy.

Risk: "Risks are future indeterminate events with a likelihood of occurrence and a possibility of loss". Risk consists of two things: the probability of something going wrong, and the negative consequences that will happen if it does. Risk identification and management are the key concerns in every software project. Efficient analysis of software risks helps in effective planning and assignments of work. Risks are identified, classified and managed before actual execution of the program takes place.

1.2 Categories of Risk

The main purpose of categorizing risk is to develop a collective viewpoint on a group of factors, that will help the managers to recognize the group that contributes maximum risk. A better way of approaching risks is to categorize them on the basis of attributes of risk. Classification of risk is an economical way of analyzing risks and their causes by grouping similar types of risks together into

classes [5]. Software risks is of two types - external and internal. The external risks come from outside of the organization and these are very difficult to control. Where as, internal risks come from the risk factors within the organization. Software risks can be classified into project risk, process risk, and product risk. This classification technique can be applied to internal risks [6], [7]. Again, risk can be categorized into three general categories [8]: project, technical, and business risk. In addition to this, risk can be categorized into performance risk, budget risk and schedule risk. [9]. In general we can say there are many risks in the software engineering. So it is very difficult to recognize all of them.

software engineering project risks are categorized on the basis of scheduling, quality, budget, and business. Since these factors can affect the risk. Some of the categories of risks are defined below,

- **Schedule Risk:** The schedule of the project gets slipped if project tasks and schedule release risks are not properly addressed. It affects on project and economy of the company and may lead to failure of the project. Schedules often slip due to the following reasons : wrong estimation of time, improper tracking of resources, failure to identify complex functionalities and time required to develop those functionalities, sudden expansion of project scope.

- **Budget Risk:** Budget risk occures if the budget of the project is not estimated properly. Budget risk occures due to the following reason: wrong estimation of budget, cost overruns, expansion of project scope.

- **Operational Risks:** This risk occurs due to improper implementation of process. Reasons for the occurance of operational risks are as follows: failure to address priority conflicts, insufficient resources, proper subject training is not done, proper resource planning is not done, communication gap in team.

- **Technical Risks:** Technical risks usually leads to failure of functionality and degradation of performance. The different causes of technical risks are as follows: continuous change in requirements, understandability of advanced technology is not available or the existing technology is in initial stages,

product is complex to implement and difficulty in integration of project modules.

While developing a test plan, risk involved in the product and the probability of occurrence of risk are taken into consideration. Along with this the damage they may cause also considered. Detailed study of the implications of risk occurrence is called risk analysis. Some other risks arise due to use of new hardware, new technology and new automation tools. Some unavoidable risk in Software Testing are:

- Frequent change in requirements.
- Incomplete requirements.
- Insufficient time for testing.
- Developers delay to deliver the build to test.
- Client needs the delivery of product urgently.
- Defect Leakage because of application size or complexity.

Some of the activities that can be done to overcome the above risks are as follows:

- Risk assessment review meeting can be conducted with the development team.
- Risk coverage profile is created by revealing the importance of each area.
- Allocate maximum resources to high risk areas and minimum resources to medium and low risk areas.
- Creating the risk assessment database which will help in maintenance in future activities and management reviews.
- Identify and define the degree of risk: high, medium and low.

1.3 Risk Assessment

The risk assessment provides information to support risk management and rank resources. Formal risk assessments are made at a later stage of development, usually when more information is available. But that will be more expensive to handle. Early stage risk assessments are carried out in the initial stages of software development. Risk assessment is of two types *qualitative* and *quantitative*. A qualitative assessment [10] approach relies more on expert knowledge, unpublished information provided by expert(s), in addition to this other available information can be considered such as observational studies or case reports. In contrast, a quantitative assessment [10] needs calculation of two factors of risk: the probability and the impact. We have considered quantitative method for our work to estimate the risk at a design level, since it provides effective means to accomplish limited resources and reduced time is required for evaluating the validity of proposals.

1.4 Motivation for Our Work

The primary motivation of our work is to overcome the drawbacks of existing risk estimation methods. The motivation behind our work is:

- To develop a risk analysis mechanism for software systems at design level using UML behavioral diagrams. For large or safety-critical systems risk analysis is very much necessary. Since failure of safety critical system leads to loss of life. So by risk analysis, we can find the risk and manage it to reduce the rate of failure and consequences of its occurrence.

- Risk analysis at early stage is beneficial since analysis at later stage is more expensive in terms of effort, cost and time. Early analysis also helps in better allocation of resources.

- To estimate risk factor at early phase of software development process using UML specifications. Since at early stage only UML diagrams are available to get the dynamic behavior of each component and connectors of the system.

It reduces the coding effort and also ensures the platform independence. This is so because the code often needs to be changed when ported to a new platform.

- To develop a risk assessment method based on quantitative metrics which can be estimated with little involvement of subjective measures from domain experts.

1.5 Objectives of Our Work

The main objective of our work is to develop a method for risk analysis at design level using UML behavioral diagrams. Our major goals are as follows:

- To estimate risk factors of a system at design level. For this we plan to :
  - Use the UML diagrams to calculate the risk factors since it is available at design level. The dynamic behavior of each component and connector can be represented by UML behavioral diagrams (state chart and sequence diagram).
  - Then concurrent control flow graph is generated to represent each scenario's control flow and for estimation of risk factor at scenario level.
  - For overall system level risk factor estimation interaction overview diagram is used.
  - Implement the proposed method.
- To analyze the sensitivity of each component and connector for each scenario and also for overall system to rank the components and connectors as per their risk factors. So allocation of resources will be easier.

1.6 Organization of the Thesis

The rest of this thesis is organized into chapters as follows.

Chapter 2 consists of the background concepts used in the rest of the thesis. First we describe some basic concepts on risk analysis, software architecture and metrics used in our proposed approach. Then, we discuss about UML diagrams (state chart, sequence and interaction overview diagram) and CCFG used for risk estimation of system along with other used formulas.

Chapter 3 presents a brief review of the related work relevant to our proposed work. We discuss the work on risk estimation of softwares at early phase of development cycle.

Chapter 4 presents our proposed work *analysis of risk at design level using UML diagrams*. We first describe the risk factor estimation method of components and connectors that consists the system. Then the scenario level risk estimation method after that the overall system level risk factor estimation technique is discussed. Finally, we present the implementation of our work along with sensitivity analysis.

Chapter 5 concludes the thesis with a summarization of our contributions. We also briefly describe the possible future extensions to our work.

Chapter 2

Basic Concepts

Chapter 2

Background

Risk Analysis is a cost-effective way to identify and manage potential problems that could undermine key business initiatives or projects. Risk consists of two things: the probability of something going wrong, and the negative consequences that will happen if it occurs.

2.1 Basic Concepts on Software Architecture

Software architecture aids as a blueprint for the system. Architecture describes the organization of a system that includes the constituting components, the relationships among them and with the development environment, and the guiding principles for its design and evolution, [IEEE 1471]. Early analysis of architecture will ensure whether the design approach will lead to an acceptable system or not. We can identify risk at design level and mitigate them by building an effective architecture. Software architecture and components are closely related. Every software systems have an architecture which can be viewed as a composition of components and connectors. A component can be defined as an abstract unit of software which provides services to other components via interface by doing some transformation of data. Transformation means performing some calculation, loading some data from secondary memory etc. A component can be a class, an object, a package of classes, or a collection of related functions. A connector is an architectural unit tasked with effecting and regulating the interaction between components. Connector facilitates communication, co-ordination, co-operation

between components. A connector can be a procedure call, pipe, data stream, client-server protocol etc. Risk assessment at the design level is more beneficial as compared to later phases of development in terms of cost and allocation of resources.

2.2 Basic Concepts on Metrics

McCabe's Cyclomatic Number

It is developed by McCabe in the year 1976 [11], to measure the complexity of a program. It measures the number of independent paths in a program. The cyclomatic complexity (cc) is calculated from a contol flow graph using the following formula:

$$CyclomaticComplexity(CC) = Numberofedges(e) - Numberofnodes(n) + 2 \tag{2.1}$$

Advantages

- It can be calculated early in the development life cycle.
- Helps in measuring the minimum effort and best areas of concentration for testing.
- Easy to apply.
- Act as a quality metric, provides relative complexity of software design.
- Helps in guiding the testing process during development by limiting the program logic .

Disadvantages

- It can measure the program complexity but not the data complexity.
- The same weight is given to both nested and non-nested loops. So deeply nested conditional structures are very difficult to understand than non-nested structures.

Information Flow Complexity Information flow complexity is introduced by Henry and Kafura in the year 1981 [12] to measure program complexity İt is defined as :

$$IF = length \times (Fan\_in \times Fan\_out)^2 \tag{2.2}$$

where length is number of lines of code, *fan_in* of a component is the number of components that call a given component and the parameters passed to it, *fan_out* of a component is the number of components called by a given component and the parameters passed from it to others. In object-oriented systems interactions among objects can be described using *fan_in* and *fan_out*. Generally high *fan_in* leads to better design of the overall system. High *fan_in* means an object is being used by other objects extensively which indicates reuse. It reduces redundant coding and makes maintainance easier. High *fan_out* indicate high degree of interdependency between classes i.e an object directly deals with a large number of other objects. High *fan_out* shows poorer design of overall system.

Advantages

- It can be applied on data-driven programs.
- It can be computed in the design phase prior to coding.

Disadvantages

- If procedures do not have any external interaction then the complexity value will be zero.

2.3 Basic Concepts on UML Diagrams

UML 2.x has 14 types of diagrams which is divided into two categories,

- Structural Diagram
- Behavioral Diagram

Structural Diagram includes seven diagrams (class, component, composite structure, deployment, object, package and profile diagram) to represent struc-

tural information. These diagrams represent the structure and they are used extensively in documenting the software architecture of software systems. Structure diagrams emphasize the things that must be present in the system being modeled.

Behavioral Diagram includes other seven diagrams out of which state machine, use case, activity diagram represent general types of behavior and other four diagrams (communication, interaction overview, timing and sequence diagram) represent different aspects of interactions . Behavioral diagrams emphasize what must happen in the system being modeled. Since behavioral diagrams explain the behavior of a system and they are used extensively for describing the functionality of software systems.

Here we have discussed the basic information about the UML diagrams which we used in our approach,

2.3.1 Use case Diagram

A use case diagram provides the higher-level view of the system. In another way, " Use case diagrams are the blueprints for your system " [13]. These diagrams provide the simplified and graphical picture of what the system actually does. It helps in gathering the requirements of a system and the factors (internal or external) which influences it. The requirements are nothing but the design requirements. So while analyzing the system to gather the functionalities use cases are set and actors are also identified. The functionalities are represented as the use cases and an actor may be a human, internal application or external application. The main purpose of use case diagrams are as follows:

- Helps in gathering requirements of a system.
- Helps to get a high-level view of a system.
- Identify external and internal factors that influences the system.
- Shows the interaction among the functionality are actors.

Use case diagrams are used in different scenarios:

- For analyzing the requirements and high-level design,
- For context modeling,
- For reverse engineering,
- For forward engineering.

Different use case dependences or relationships are include, extend, generalization, specialization. These are defined as follows,

Include - An include relationship of a use case (base use case) includes the functionality of another use case (the inclusion use case). It supports the reuse of functionality in a use case model.

We can use include relationships in following situations:

- The feature of the inclusion use case is same for two or more use cases
- The outcome of the behavior which the inclusion use case specifies is significant to the base use case.

The Fig 2.1 demonstrates an e-commerce application that offers customers with an alternative of checking the status of their orders. The checkOrderStatus use case represents this behavior. It has an inclusion use case named logIn. The logIn use case is a distinct inclusion use case as it includes behaviors that several other use cases in the system use. An include relationship points from the checkOrderStatus use case to the logIn use case is to specify that the checkOrderStatus use case constantly includes the behaviors in the logIn use case.

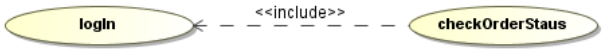


Figure 2.1: Use case for include relationship

Extend - An extend relationship of one use case (extension) extends the features of another use case (base). It helps to reveal the details about a system or application which are typically hidden in a use case.

The extend relationship states that the inclusion of the extension use case is dependent on what happens when the base use case executes. We can indicate numerous extend relationships for a single base use case.

The extension use case is not meaningful on its own whereas the base use case is defined independently and is meaningful by itself. The extension use case comprises of one or more behavior sequences that explain supplementary behavior which can incrementally add to the behavior of the base use case. Every segment can be added into the base use case at a different point, called an extension point.

The features of the base use case can be accessed and changed by extension use case. But, the base use case cannot access or change the attributes and operations of the extension use case.

We can append extend relationships to a model to illustrate the following circumstances:

- A fraction of a use case which is optional system behavior.
- A subflow is executed only under definite circumstances.
- A set of behavior that may be introduced in a base use case.

The following Fig 2.2 of e-commerce system shows base a use case named placeOrder which has an extending use case named shippingDetail. An extend relationship spots from the shippingDetail use case to the placeOrder use case to specify that the behaviors in the shippingDetail use case are optional and occur only in definite conditions.

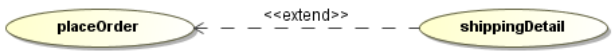


Figure 2.2: Use case for extend relationship

Generalization - If two or more use cases have some common features in behavior, structure, and purpose then we can use generalization. It describes the common features in a new use case.

Specialization - A base use case may be specialized into one or more derived use cases. These derived use cases represents more specific features of the base use case. Both the base use case and derived use case need not to be necessarily abstract, although in most cases the base use case is abstract. A derived use case inherits all behavior, structure, and relationships of the base use case. Derived use cases of the same base use case are all specializations of that base use case.

2.3.2 State chart Diagram

It is one of UML diagrams which shows the dynamic behavior of a system in response to internal or external stimuli. Then the behavior is explored and represented in a sequence of events, that could take place in one or more probable states. The dynamic flow of control from one state to another is shown by state chart diagram. Each state chart diagram typically represent objects of an individual class and keeps track of various states of its objects over the system. Here we can say it models the life span of an object from its formation to end. The main purpose of state chart diagram is to model reactive systems. Reactive system means, a system which responds to an internal and external stimulus.
The state chart diagram is used for the following purposes:

- For modeling dynamic aspect of a system.
- For modeling life span of a reactive system.
- For describing various states of an object in its life time.
- To define a state machine which model states of an object.
- To recognize events accountable/responsible for state changes.
- For forward and reverse engineering.

2.3.3 Sequence Diagram

Sequence diagram is one of the interaction diagram which represents how processes interact with each other and in what sequence. It helps in dynamic modeling of

the system which focuses on identifying the behavior of the system. It shows the sequence of messages transferred between different objects. It shows the objects and classes involved in a particular scenario and the order of messages exchanged between them to do the functionality of that scenario. Each use case functionality can be realized using sequence diagram to get the logical view of the system under development.

A sequence diagram consists of parallel vertical lines and horizontal arrows. Where parallel vertical lines represent lifelines of objects, the horizontal arrows represent messages exchanged between them in the sequence in which they occur. It models the flow of logic inside the system in a visual manner. It helps in validating and documenting the logic as well as used for analysis and design purposes.

Sequence diagrams are used for the following purposes:

- Modeling usage scenarios.
- Modeling the logic of methods.
- Modeling the logic of services.

2.3.4 Interaction Overview Diagram

Interaction overview diagram is one of the fourteen diagrams of UML. It pictures the control flow with nodes that can contain sequence diagram, timing diagram and communication diagram. It shows the sequence of activities like activity digram. But the difference is that each activity is pictured as a frame which can contain other diagrams like sequence diagram, communication diagram, timing diagram. With these elements the interaction overview diagram can be used to deconstruct a complex scenario that would otherwise require multiple if-then-else paths to be illustrated as a single sequence diagram. In other way, we can say it provides an overview of the relationship between two more specialized UML diagrams like sequence diagrams, communication diagrams, timing diagrams

2.4 Concurrent Control Flow Graph

Generally CCFG is useful to represent the concurrent control flows in a program. In concurrent computation the programs are designed as group of interacting computational units which may be parallelly executed. Concurrent programs or threads can be executed on a single processor or in parallel. Execution on a single processor is done by interleaving the execution steps of each in a time-slicing method. Execution in parallel is possible by allocating each computational process to one of a set of processors which is closed or distributed through a network.

In design level also we can use CCFGs to analyze the concurrent control flow of sequence diagrams [14]. Since conventional Control Flow Analysis (CFA) [15] methods are usually applied to sequential programs. Sequential program means there is no concurrency in a single module. The most commonly used control flow model is Control Flow Graph (CFG) [15]. But, the use of the CFG model is restricted to only sequential programs. Its standard model cannot be easily used to accomplish control flow analysis in a module if intra-module concurrency is there, or some of the statements run parallelly with others.

From the UML 2.x sequence diagram metamodel [14] it is very clear that asynchronous messages and par interaction operator require intra-sequence diagram concurrency. But, such concurrency cannot be explored by conventional CFGs. The above two modeling features which lead to Concurrency has to be taken into consideration while analyzing the control flow in sequence diagram. The impacts of these two modeling features are discussed below,

Impact of Asynchronous Messages: UML introduces two types of messages for sequence diagrams: synchronous and asynchronous.

Synchronous message - Here the caller waits for completion of the invoked behavior and expects a return values.

Asynchronous message - Here the caller proceeds immediately and does not expect a return value.

Hence, asynchronous messages of a sequence diagram will involve a concurrent control flow in the sequence diagram. For better understandability of the impact

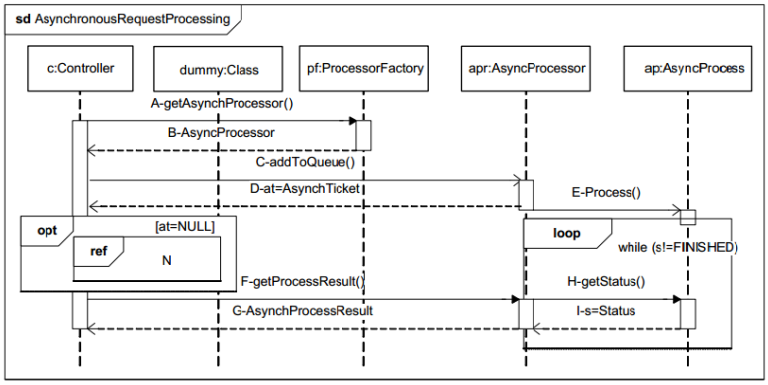


Figure 2.3: A sequence diagram with asynchronous messages

in CFA due to occurrence of asynchronous messages, we have considered an example. The sequence diagram shown in Fig 2.3 describes an asynchronous request processing approach.

Message are prefixed with alphabetical symbols for easier reference to the messages. The sequence diagram has both synchronous and asynchronous messages. There are five objects such as c, apr, dummy, pf and ap. To realize the effect of asynchronous messages of the sequence diagram in the control flow, let us consider the case when the message *addToQueue()* is send by the object c of class Controller to the object apr of class *AsyncProcessor*. As *addToQueue()* message is asynchronous, the caller will not wait for the reply message of the asynchCall message *addToQueue()*. It will continue running the other messages in its lifeline. In the mean while, object apr will start executing *addToQueue()* method. So we can say, in addition to the thread of control for object c, a new concurrent thread of control will be created for object apr. Hence, we can say asynchronous messages has impact on control flow of a sequence diagram. Since it introduces two separate concurrent threads of control, one for the sender object and another for the receiver object of a message [14].

Impact of par Interaction Operator: Another new feature introduced by UML 2.x is par interaction operator. It supports parallel execution of interaction fragments. A sequence diagram where interaction operator is used is shown in the Fig 2.4. A *cook* use case of a microwave oven is defined by the sequence dia-

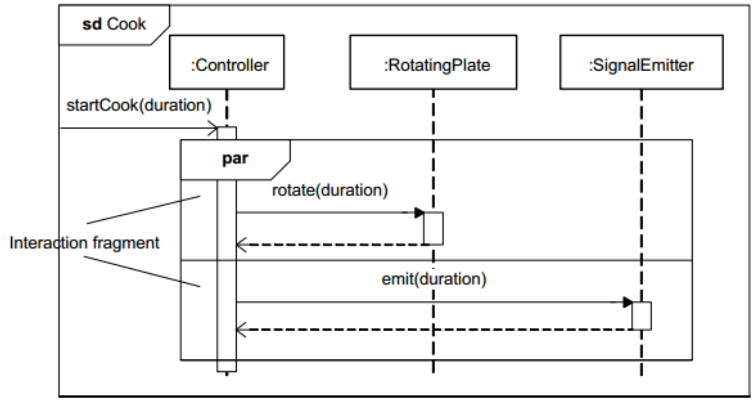


Figure 2.4: A sequence diagram with par interaction operator

gram. The controller of the oven is the object *Controller*. Other two objects are *RotatingPlate* and *SignalEmitter*. The controller receives a *startCook(duration)* message, where *duration* tells about the time required to cook. Then, the controller parallelly sends two messages, one to *RotatingPlate* and another one to *SignalEmitter* to start doing their tasks. So two interaction fragments holding message *rotate(duration)* and *emit(duration)* run concurrently. Hence, par interaction operators also impact the control flow of a sequence diagram by involving two or more concurrent threads of control [14].

So concurrent control flow graph is needed to describe the control flow of a sequence diagram. For each sequence diagram a CCFG will be generated. There is some sequence diagrams which calls other sequence diagrams, in that scenario a control flow edge will connect the corresponding CCFG of sequence diagrams to generate the inter-sequence diagram CCFG. The inter-sequence diagram CCFG is similar to the idea of interprocedural CFG [15]. The mapping rules from sequence diagram (SD) to concurrent control flow graph (CCFG) features are shown in Table 2.1.

Table 2.1: Mapping rules from SD features to CCFG features

Rule Number	Sequence Diagram Features	Concurrent Control Flow Graph Features
1	InteractionFragment	Activity
2	First message end	Flow between InitialNode and first control node
3	SynchCall/SynchSignal	CallNode
4	AsynchCall or AsynchSignal	(CallNode + ForkNode) or ReplyNode
5	Message SendEvent and message ReceiveEvent	ControlFlow
6	Lifeline	ObjectPartition
7	par CombinedFragment	ForkNode
8	loop CombinedFragment	DecisionNode
9	alt/opt CombinedFragment	DecisionNode
10	break CombinedFragment	ActivityEdge
11	Last message ends	Flows between ending control nodes and ActivityFinalNode
12	InteractionOccurrence	Control Flow across CCFGs
13	Polymorphic message	DecisionNode
14	Nested InteractionFragments	Nested CCFGs

Chapter 3

Related Work

Chapter 3

Review of Related Work

This chapter presents an overview of the research work carried out related to our research work.

A number of metrics have been developed to measure the design quality of a object-oriented software [16], [17]. Design metrics are categorized as static and dynamic. A quality metric should associate external quality attributes (maintainability, reusability, understandability, error-proneness) of a design [18]. On the basis of observation and empirical studies it was found that coupling [19] and complexity [20] have direct impact on quality of software. Based on this Yacoub et al. [21] introduced a metrics suite to measure the design qualities at early development stage. This metric suite includes metrics for object coupling and dynamic complexity based on execution scenario. Here we are focusing on this metrics to calculate the risk factors of different components and connectors.

Almendros-Jimenez and Iribarne [22] explained a method to describe use cases by means of sequence diagrams. They also compares sequence diagrams to define sequence diagram relationships for recognizing and describing use case relationships. Their main aim is to provide a semantics of use case relationships by means of sequence diagrams. They described how to map each relationship (generalization, specialization, extend, include) of use case to the relationship of sequence diagram. We followed their method to get the sequence diagram from use case diagram for our approach.

Garousi et al. [14] presents a methodology for control flow analysis on the basis of UML diagrams. Generally control flow analysis is done at code level but

this method can be applied at early phase of software development life cycle. It uses UML diagrams for control flow analysis. They proposed a extended activity metamodel kown as concurrent control flow graph (CCFG) for control flow analysis. They define an OCL-based mapping rules between a sequence diagram and a CCFG. They also define Concurrent Control Flow Paths, which are a generalization of the conventional Control Flow Path concept. We have followed their technique of mapping from sequence diagram to concurrent control flow graph.

Jeffrey D. Gordon [23] proposed three distinct UML methods to compute effort corresponding to early, mid and late stage of design phase. In early stage he used object point to compute the effort. In mid stage interaction point is used to calculate the effort. For late stage of design message pont and transaction point is used to calculate the effort of software. Here we have used the interaction point method for our work.

Amland [24] proposed a method for risk-based testing where risk is estimated for a high level function on the basis of failure probability and cost of failure of that fuction. Failure probability is estimated on the basis of four factors : new functionality, size, design quality and complexity. The cost of failure is calculated based on both customer and supplier cost. The main drawback of this approach is the calculation of complexity. Since it considers informal way to calculate which is based on subjective judgment of domain experts.

Different risk assessment methods [25], [26] are developed at requirement phase on the basis of experts knowledge. These two approaches initially identify different modes of failures for high-level requirement and then attempt to estimate the effect of these failures on the requirement. It does not take into account any structural and behavioral dependencies between interacting objects of the system. So it is purely subjective and more human intensive.

Several methods for reliability-based risk assessment on the basis of formal design model [27], [28] are available. Yacoub and Ammar [27] developed a component-based risk assessment method using UML diagrams. They calculated the heuristic risk factor of each component and connector individually. Then, the overall sys-

tem level risk factor is estimated using the component and connector risk factor along with the transition probabilities between components and connectors. For estimatinig the overall system level risk factor they developed an intermediate graph known as Component Dependence Graph (CDG) on the basis of scenarios.

A similar type of approach is introduced by Goseva-Popstojanova et al. [28] for risk assessment. They also calculated individual component and connector risk factor. Then used these values along with Discrete Time Markove Chain and transition probability to estimate the scenario level risk factor. Multi-failure states also introduced by them for each scenario which represents the failure modes with different severity level.

Appukkutyet al. [29] introduced a risk assessment method which considers possible failure modes of a scenario and computed the complexity of the scenario in each failure mode. This method is for risk assessment at requirement phase. So, the risk related with low-level details (component and connector level details) are not taken into cosideration.

For estimating the risk factor probability of failure and consequences of its occurance is taken in to consideration. Luke [30] Software failure probability is difficult to find in advance, design complexity is linearly related to rate of defect. So occurance of defects at code level should be calculated using cyclomatic or Halsteads complexity. At design level we can estimate this complexity using UML diagrams.

Cortesselaet al. [31] have also introduced a risk assessment method using UML diagrams. This approach estimates performance-based risk. This method uses UML diagrams to estimate the performance failure probability. Then combines it with the failure severity which is computed using the Functional Failure Analysis. Which helps to determine risky scenarios along with risky software components.

Cheung [32] proposed a user-oriented reliability model to measure the reliability of a software system with respect to a user environment. A simple Markov model is formulated to determine the reliability of a software system on the basis of reliability of each individual module and the measured inter-modular transition

probabilities as the user profile. Sensitivity analysis techniques are developed to determine modules most critical to system reliability.

Chapter 4

Proposed Work

Chapter 4

Analysis of Risk at Design Level using UML Diagrams

In this chapter, we introduce our approach for risk assessment [32]. We first describe the risk analysis process. Then, we discuss the risk calculation techniques of components and connectors for each scenario. Next, we present the scenario level and overall system level risk calculation techniques along with sensitivity analysis.

4.1 Basic Concepts

Information Flow Complexity: Information flow complexity is used to measure program complexity.It is defined as :

$$IF = length \times (Fan\_in \times Fan\_out)^2 \tag{4.1}$$

where length is number of lines of code, fan_in is the number of components called this component and the parameters passed to it, fan_out is the number of components called by this component and the parameters passed from it to others. It can also be applied at design level where length is the cyclomatic complexity.

Cyclomatic Complexity: It is introduced by McCabe in the year 1976, used to measure the complexity of a program. It measures the number of independent paths in a program . The cyclomatic complexity (CC) is calculated from a control

flow graph using the following formula:

$$CyclomaticComplexity(CC) = Numberofedges(e) - Numberofnodes(n) + 2 \tag{4.2}$$

4.2 Case study

Here we have taken the *Library Management System (LMS)* case study for our research work. There are four components in this as user, librarian, member record(MR), book. User component can issue book, renew book, return book, reserve book. Librarian manages user, book and member record. Member record manages member information and their requests. Book information is handled by the component book. Also, we have used six connectors for communication between components. The connectors between different components are as follows,

(U_L) - Connector between component user and librarian

(L_U) - Connector between component librarian and user

(L_M) - Connector between component librarian and member record

(L_B) - Connector between component librarian and book

(M_L) - Connector between component member record and librarian

(B_L) - Connector between component book and librarian

The usecase diagram is shown in the Fig 4.1

4.3 Our approach for Risk analysis

In our approach we have considered use case diagram for analyzing the risk of a system. Each use case in the use case diagram represents some high-level functionality. Each use case can represent one or more scenario. Each scenario is realized using sequence diagrams as shown in Fig 4.3. Sequence diagram represents how the components of a scenario are interacting with each other and the order of message exchanged among them. For each scenario, we have calculated the risk factors of components/connectors. The risk factors are calculated as a product of the dynamic complexity/coupling and the severity value. These severity values

are assigned to the component/connector by domain experts using hazard analysis and failure mode effect analysis. Then we construct concurrent control flow graph (CCFG) for each scenario according to the sequence diagram to estimate the risk factor of each scenario. The overall system level risk factor is estimated using an interaction overview diagram. The algorithm for risk analysis is defined below,

Algorithm 1 The risk analysis process

```
for each use case do
  for each scenario do
    for each component do
      Calculate the dynamic complexity using information flow complexity
      Assign severity based on FEMA and hazard analysis
      Estimate the risk factor of component
    end for
    for each connector do
      Calculate the dynamic Coupling
      Assign severity based on FEMA and hazard analysis
      Estimate the risk factor of connector
    end for
    Construct the Concurrent control flow graph
    Calculate risk factor of scenario
  end for
end for
Construct interaction overview diagram and calculate the transition probability
Calculate the overall system level risk factor
Do sensitivity analysis to find most sensitive components, connectors and scenarios
```

4.4 Risk Factor Assessment of Components/ Connectors

We calculate the risk factor of each component and connector involved in each scenario S_x as a product of the dynamic complexity/coupling and the severity values of those component and connector. Severity value varies from scenario to scenario.

For each component i, involved in scenario S_x, risk factor rsf_i^x is calculated as

$$rsf_i^x = DCm_i^x \times svrt_i^x \tag{4.3}$$

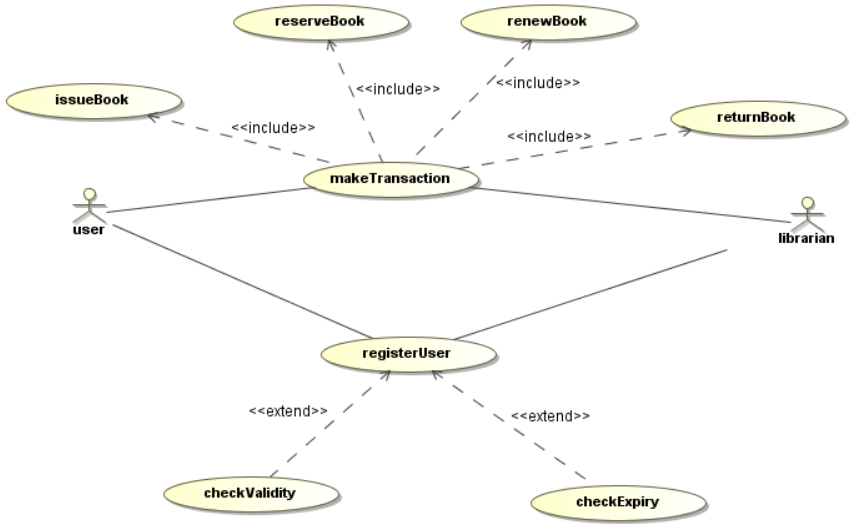


Figure 4.1: Usecase diagram of Library Management System

where $DCm_i^x (0 \leq DCm_i^x < 1)$ is the normalized dynamic complexity and $svrt_i^x (0 \leq svrt_i^x < 1)$ is the severity level of component i in that particular scenario S_x .

For each connector between two components i and j, involved in scenario S_x, the risk rsf_{ij}^x is calculated as

$$rsf_{ij}^x = DCn_i^x \times svrt_i^x \tag{4.4}$$

where $DCn_i^x (0 \leq DCn_i^x < 1)$ is the normalized dynamic coupling and $svrt_i^x (0 \leq svrt_i^x < 1)$ is the severity level of connector between two components i and j involved in scenario S_x.

Then we describe the methods to estimate the normalized dynamic complexity DCm_i^x and normalized dynamic coupling DCn_i^x for component and connector respectively along with the severity calculation for component and connector.

4.4.1 Calculation of Normalized Dynamic Complexity

Information flow complexity (IF) is introduced by Henry and Kafura in the year 1981 [12] to measure program complexity. It can also be applied at design level, and it is defined as :

$$IF = length \times (Fan\_in \times Fan\_out)^2 \tag{4.5}$$

where length is the cyclomatic complexity, *fan_in* is the number of components called this component and the parameters passed to it, *fan_out* is the number of components called by this component and the parameters passed from it to others. Cyclomatic complexity (CC) is introduced by McCabe in 1976, it is defined as $CC = e - n + 2$ and calculated from the control flow graph. Where e is the number of edges and n is the number of nodes. Instead of using control flow graph to calculate CC, we have used state chart diagram of components. State chart diagram of a component is considered for each scenario which is available at design level. For each component i the state chart diagram represent the number of states and transitions among these states which describes the dynamic behavior of that component. For a component i a subset of all its states and corresponding transitions are traversed for each scenario. Let the subset of states traversed by a component i in the scenario S_x is denoted as P_i^x and the subset of transitions traversed by a component i in the scenario S_x is denoted as Q_i^x . Then mapping of the subset of states traversed P_i^x and corresponding transitions Q_i^x to a control flow graph is done. Where the number of nodes $p_i^x = |P_i^x|$ (cardinality of P_i^x) and the number of transitions $q_i^x = |Q_i^x|$ (cardinality of Q_i^x). Here the cyclomatic complexity for a component i in the scenario S_x is defined as,

$$CC_i^x = q_i^x - p_i^x + 2 \tag{4.6}$$

Then, we use sequence diagrams of components for each scenario S_x to estimate the *fan_in* and *fan_out*. For a component i in a scenario S_x *fan_in* FI_i^x is defined as the sum of messages and parameters transmitted to it. Whereas *fan_out* FO_i^x is the sum of messages and parameters transmitted from it. Here, we applied IF at the design level to estimate the dynamic complexity of components. The dynamic complexity of component i in the scenario S_x is defined as,

$$dcm_i^x = CC_i^x \times (FI_i^x \times FO_i^x)^2 \tag{4.7}$$

For a component i in the scenario S_x the normalized dynamic complexity DCm_i^x is estimated as,

$$DCm_i{}^x = \frac{dcm_i{}^x}{\sum_{k \epsilon S_x} dcm_i{}^x} \tag{4.8}$$

Table 4.1: Normalized dynamic complexity values of components for *issueBook* scenario

Component Name	Normalized dynamic complexity
User	0.1463
Librarian	0.8468
MR	0.0013
Book	0.0054

The state chart diagram of component *user* in the scenario *issueBook* and the corresponding sequence diagram are shown in the Fig 4.2 and Fig 4.3 respectively which are used to calculate the dynamic complexity using eq(4.7). Then the normalized dynamic complexity is calculated using eq(4.8) and shown in Table 4.1

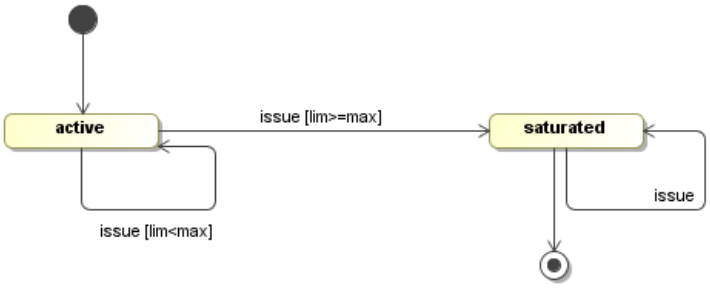


Figure 4.2: State chart diagram of component *user* in *issueBook* scenario

4.4.2 Calculation of Normalized Dynamic Coupling

Here we use sequence diagram to estimate the dynamic coupling of connectors by applying the formulas mentioned in [10]. The dynamic coupling of a connector is defined as follows,

$$DCn = \frac{|MS_{ij}^{x}|}{|MS^{x}|} \tag{4.9}$$

where MS_{ij}^{x} represents the set of messages transmitted from component i to component j in the scenario S_x and MS^x represents set of all messages transmitted among all active components in that scenario S_x. Then the normalized dynamic coupling for *issueBook* scenario using the sequence diagram shown in Fig 4.3 is calculated using eq(4.9) and shown in Table 4.2

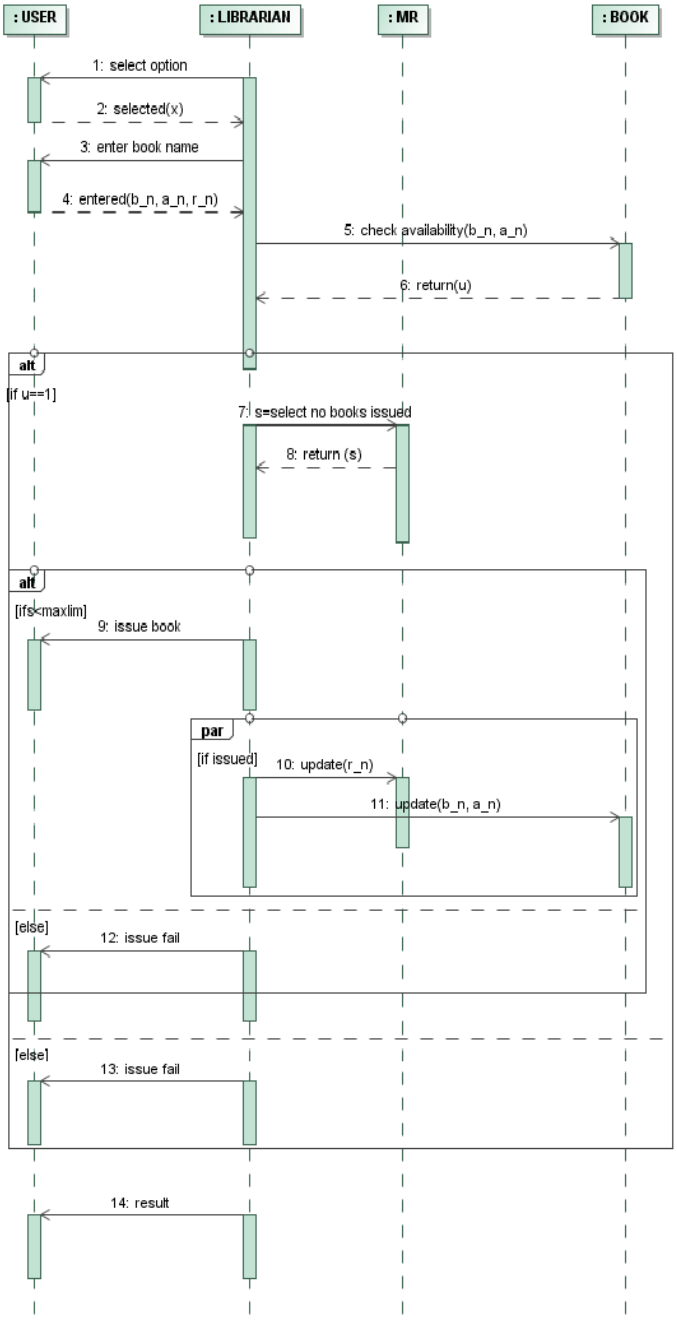


Figure 4.3: Sequence diagram of *issueBook* scenario

Table 4.2: Normalized dynamic coupling values of connectors for *issueBook* scenario

Connector Name	Normalized Dynamic Coupling
U_L	0.1428
L_U	0.4285
L_M	0.1428
L_B	0.1428
M_L	0.7142
B_L	0.7142

4.5 Hazard and Severity Analysis

Here we find out the severity of failure of each component and connector for each scenario to calculate the risk factors of component and connector. Severity or hazard analysis is the most important step of risk assessment. Hazard is nothing but the consequences of happening of some undesired events and severity is the seriousness of that hazard. It is very important because sometimes the complexity may be very less but the severity of failure is very high. So we have considered the severity of failure of each component and connector on the basis of their failure consequences. For estimating the severity we need domain experts, they rank the severity of failure of each component and connector on the basis of their domain knowledge and experience. According to MIL_STD_1629A, severity takes into consideration the worst-case consequence of a failure obtained by the degree of injury, system damage, property damage, and mission loss that could eventually occur. On the basis of hazard analysis [33], we find out the following severity classes. The first one is catastrophic, which is defined as a failure that may result death or entire system loss. Next is critical, which is defined as a failure that may result severe injury in terms of major property loss or major system loss. Third one is marginal means a failure may result minor injury in terms of minor property loss, minor system loss or delay. Last one is minor, defined as a failure which may result in any type of damage but leads to an unscheduled maintenance or repair. Each severity class is assigned with a severity value like 0.25 for minor, 0.5 for marginal, 0.75 for critical, 0.95 for catastrophic [28]. On the basis of study performed by Ammar et al. [34] the severity values for each severity class are taken

Table 4.3: Analysis of severity for components in the *issueBook* scenario

Component	Accident	Severity
User	User component malfunctioning	Minor
Librarian	Interface failure, communication failure between components	Marginal
MR	Database failure	Critical
Book	Database failure	Critical

Table 4.4: Analysis of severity for connectors in the *issueBook* scenario

Connector	Accident	Severity
U_L	Transmits incorrect command, error message is received	Minor
L_U	Transmits incorrect command	Marginal
L_M	Transmits incorrect command,Component MR fails to update	Critical
L_B	Transmits incorrect command, Component book fails to update	Critical
M_L	Transmits incorrect command	Marginal
B_L	Transmits incorrect command, message received is errorneous	Minor

in a linear scale. The severity values of components for the issue book scenario are shown in Table 4.3 and connectors are shown in Table 4.4. Then the risk factor of each component and connector as a product of dynamic complexity/coupling and severity level. The risk factor of each component and connector for issueBook scenario are shown in Table 4.5 and Table 4.6 respectively.

4.6 Estimation of Scenario Level Risk factor

We have introduced an analytical estimation approach for calculation of scenario level risk factor using CCFG. We considered the failure of both component and connector in contrast to [32] where only component failure is taken into consideration. A component or connector can fail at any point of time during the execution of a scenario. Due to this, we have considered the risk factor of each component and connector after every step of execution in the scenario while calculating the

Table 4.5: Risk factor of components of *issueBook* scenario

Component Name	Severity	Risk factor
User	0.25	0.0363
Librarian	0.5	0.4211
MR	0.75	0.00101
Book	0.75	0.0040

Table 4.6: Risk factor of connectors of *issueBook* scenario

Connector Name	Severity	Risk Factor
U_L	0.25	0.0357
L_U	0.5	0.2142
L_M	0.75	0.1071
L_B	0.75	0.1071
M_L	0.5	0.0357
B_L	0.25	0.0178

Table 4.7: Component and connector involved in each node in CCFG of *issueBook* scenario

Node Name	Sender Component	Receiver Component	Connector
l_u	Librarian	User	L_U
u_l	User	Librarian	U_L
l_u	Librarian	User	L_U
u_l	User	Librarian	U_L
l_b	Librarian	Book	L_B
b_l	Book	Librarian	B_L
l_m	Librarian	MR	L_M
m_l	MR	Librarian	M_L
l_u	Librarian	User	L_U
l_m	Librarian	MR	L_M
l_b	Librarian	Book	L_B
l_u	Librarian	User	L_U
l_u	Librarian	User	L_U
l_u	Librarian	User	L_U

scenario level risk factor.

For each scenario, CCFG is modeled from the sequence diagram of that scenario [14]. In CCFG, each node represents the message transmitted between the components. CCFG is used to analyze the concurrent flow in the sequence diagram. The CCFG of *issueBook* scenario is shown in the Fig 4.4. We used a table to keep track of the sender and receiver component and the connector between them for each message as shown in Table 4.7. We then calculated the risk factor at each stage of execution by taking the risk factors of component and connector. By using the following formula, we have calculated the risk factor (M_i) of each scenario.

$$M_i = \sum_{j=1}^{nop} \sum_{i=1}^{n} R_i = \sum_{j=1}^{nop} (R_1 + R_2 + R_3 + ...) \tag{4.10}$$

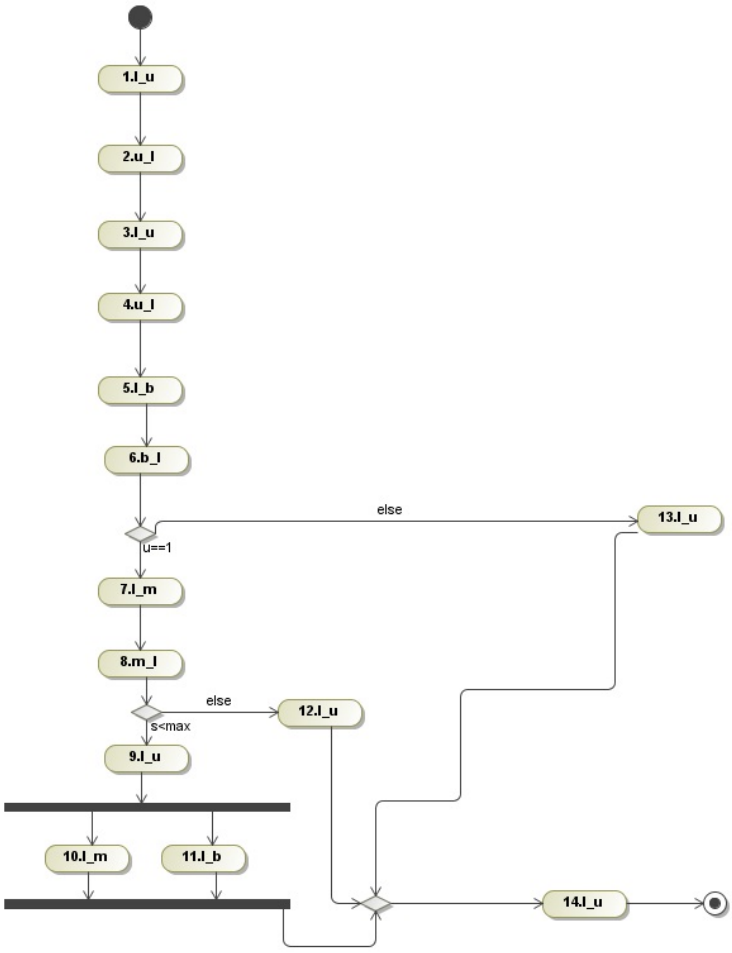


Figure 4.4: Concurrent control flow graph of *issueBook* scenario

Table 4.8: Normalized risk factor of each scenario

Name of Scenario	Risk Factor	Normalized Risk Factor
issue_book	2.9217	0.3072
Renew_book	0.9892	0.1040
Reserve_book	2.9221	0.3073
Return_book	0.9583	0.1007
invalid_card	0.7955	0.0836
valid_card	0.9217	0.0969

where,

$$R_i = R(T_i) + (1 - R(T_i)) \times R(T_i, D_i) + (1 - R(T_i)) \times (1 - R(T_i, D_i)) \times R(D_i)$$

In this case nop represents the number of paths in CCFG

n represents the number of nodes in a path

$R(T_i)$ represents risk factor when sender component T_i fails (calculated using eq(4.3))

$(1 - R(T_i)) \times R(T_i, D_i)$ represents risk factor when connector between sender component Ti and receiver component Di fails (calculated using eq(4.4))

$(1 - R(T_i)) \times (1 - R(T_i, D_i)) \times R(D_i)$ represents risk factor when receiver component D_i fails.

Then normalized risk factor of each scenario is calculated as follows,

$$NM_i = \frac{M_i}{\sum_{i=1}^{ns} M_i} \tag{4.11}$$

Where ns represents the number of scenarios in the system. The normalized risk factor of each scenario is shown in Table 4.8

4.7 Estimation of System Level Risk Factor

For estimating the overall system level risk factor, we use the *scenario-based specifications*. Scenario specification is the collection of a set of scenarios [35] , [36]. To describe the behavior of an application, scenario specification is extensively used in industry. Scenario specification can be modeled through *Interaction Overview*

Diagram (IOD) [35]. It pictures the cooperation among different interaction diagrams, as its nodes, to elucidate a control flow between them in terms of logic and process-flow. In other way, we can say it provides an overview of the relationship between two more specialized UML diagrams like sequence diagrams, communication diagrams, timing diagrams. Here, we have taken the sequence diagram as nodes which are placed in a specific order. The transition from one node to another has some transition probability TP_{ij} which represents the scenario j will be executed after executing the scenario i. We have used the operational profile [22] to find the transition probability. System reliability is affected by the reliability of components and the transition probability of scenarios [35]. On the basis of this we estimated the risk factor of the overall system as follows:

$$R\left(Sym\right) = \sum_{m=1}^{nop} \sum_{j=1}^{nod} NM_j \times T_{ij} \tag{4.12}$$

The overall system risk factor is 0.513438. The *IOD* of LMS system is shown in Fig 4.5.

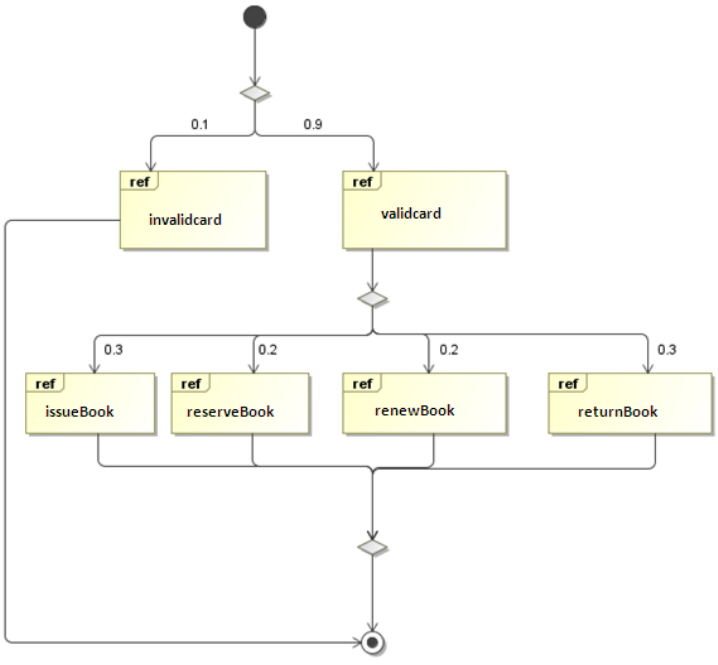


Figure 4.5: Interaction overview diagram *(IOD)* of Library Management System

4.8 Implementation

In this section, we discuss our implemented approach and prove its correctness. We have used Java to implement our proposed method. First, we draw the state chart and sequence diagrams using MagicDraw [37]. Next, we export the xml file of the diagram which is given as input to our program. As an outcome, we obtain the normalized risk factors of the components and connectors. The normalized risk factors of the components and connectors are shown as in Fig 4.6 and Fig 4.7, respectively.

In the next step, we used the result obtained in the previous step to calculate the scenario level risk factor. The normalized risk factors of the scenarios of LMS are shown in Fig 4.8

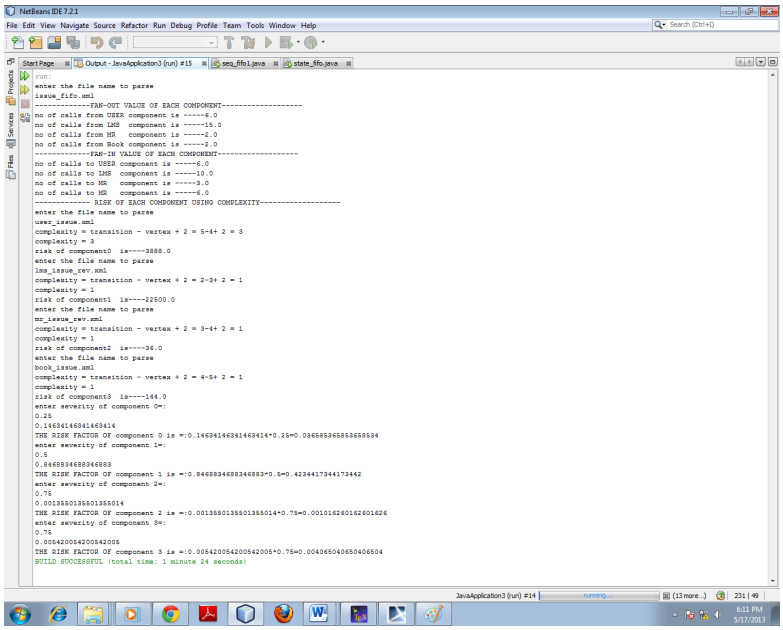


Figure 4.6: Normalized risk factor of each component for *issueBook scenario*

4.8.1 Sensitivity Analysis

Sensitivity analysis is a simple and effective way to calculate the uncertainty in results obtained for a system. It tests the robustness of the results obtained while there is an uncertainty in the inputs. Fig 4.9 shows the sensitivity of overall system risk to the risk factors of the components. The component librarian, MR and user

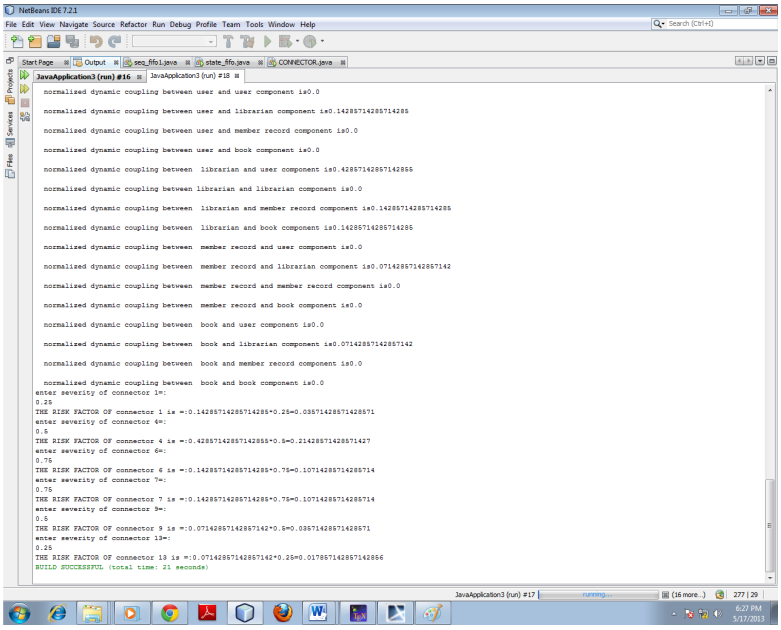
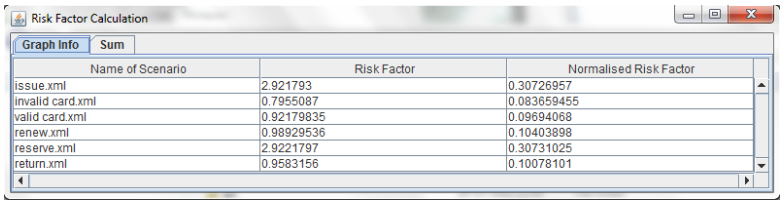


Figure 4.7: Normalized risk factor of each connector for *issueBook scenario*



Name of Scenario	Risk Factor	Normalised Risk Factor
issue.xml	2.921793	0.30726957
invalid card.xml	0.7955087	0.083659455
valid card.xml	0.92179835	0.09694068
renew.xml	0.98929536	0.10403898
reserve.xml	2.9221797	0.30731025
return.xml	0.9583156	0.10078101

Figure 4.8: Normalized risk factor of each scenario

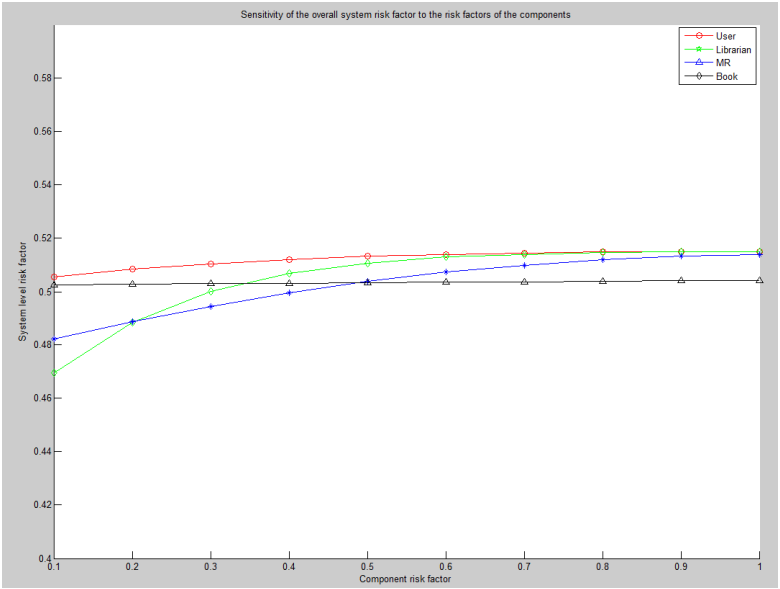


Figure 4.9: Sensitivity of overall system risk factor to risk factors of components

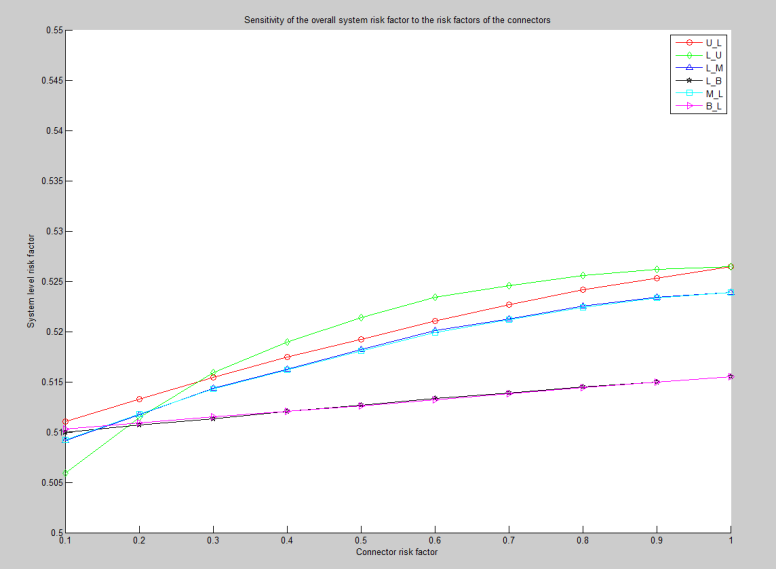


Figure 4.10: Sensitivity of overall system risk factor to risk factors of connectors

are almost certainly affecting the overall system risk because these components are having high execution probabilities in all scenarios. The book component is less likely to affect the overall risk of the system. In Fig 4.10, we have shown the sensitivity of overall system risk to the risk factors of the connectors involved. The connectors L_M, M_L,U_L and L_U have most influence on the risk of the overall system as they are the mostly used connectors in the scenario. Similarly, we have done sensitivity analysis for components and connectors in different scenarios.

4.9 Comparison with Related Work

Popstojanova et al. [28] introduced a methodology to estimate the risk factor of the overall system using dynamic complexity of component and connector. To compute the dynamic complexity of a component they have considered cyclomatic complexity of the component which does not consider the data flow complexity. But in our case, we used the information flow metric which considers data flow complexity. For scenario level risk estimation they used Markov model. And they considered the failure of components/connectors only once during the execution of a scenario. But in our case we have taken all possible failure of components/connectors during the execution of each step of a scenario using CCFG.

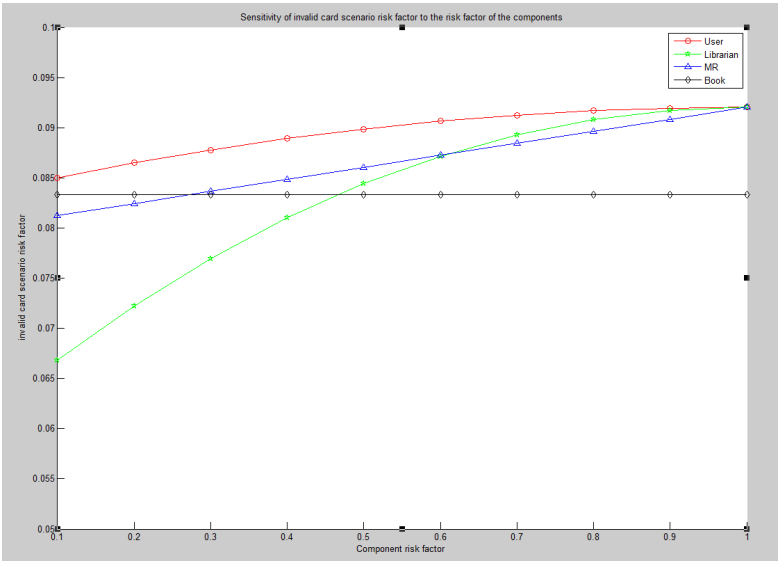


Figure 4.11: Sensitivity of invalid card scenario risk factor to risk factors of connectors

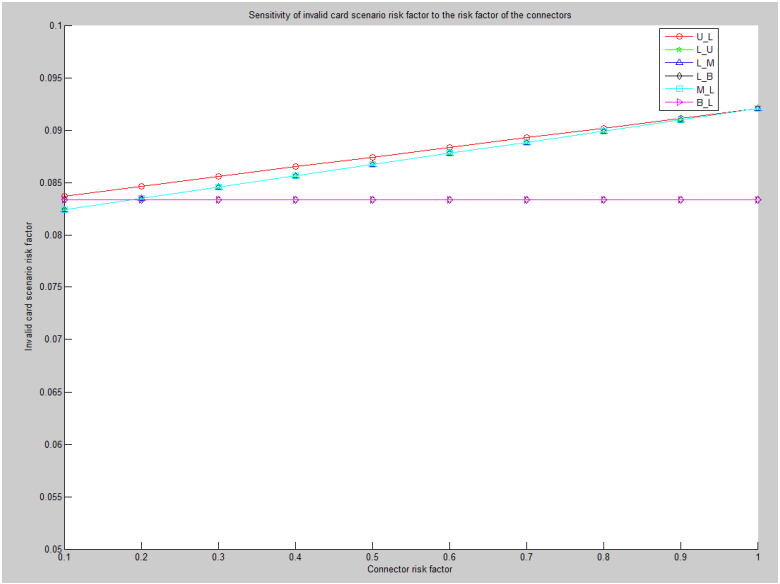


Figure 4.12: Sensitivity of invalid card scenario risk factor to risk factors of components

Along with this in our approach parallel execution is also considered. In our approach in addition to this different use case dependencies are also considered [22].

Yacoub and Ammar [27] used the severity and complexity to estimate the risk factor of component and connector. They used component dependence graph for each scenario to represent the component, connector and transition probability. Then overall system risk factor is estimated using the risk factor of consisting components and connectors and the the aggregation algorithm. But they have not considered the scenario level risk factor which we have estimated using CCFG. By this, we can also find the scenario that is of high-risk.

Sadi et al [38] proposed a method to minimize the risks of soft errors in mobile and ubiquitous systems, where they only considered the criticality of components. But in our case, we have considered both component and connector risk factors. Connector is responsible for transmission of any information. So the system will be affected if it fails.

Cheung [32] proposed a user-oriented reliability model to measure the reliability of a software system with respect to a user environment. A simple Markov model is formulated to determine the reliability of a software system on the basis of reliability of each individual module and the measured inter-modular transition probabilities as the user profile. Sensitivity analysis techniques are developed to determine modules most critical to system reliability. They have considered only component reliability while calculating the system level reliability. But in our approach we have considered both component and connector risk factor while calculating system level risk factor.

Chapter 5

Conclusion

Chapter 5

Conclusion

We have proposed a method for early assessment of risk using UML diagrams like use case, state chart, sequence diagram and interaction overview diagrams. Initially we estimated the risk factor of component and connector using state chart diagram and sequence diagram. Then *CCFG* is used for scenario level risk factor estimation. Finally the system level risk factor is estimated using scenario level risk factor and transition probability of scenario. We have used the operational profile of the system to know the transition probability. Next, we have done the sensitivity analysis of component and connector risk factors with respect to scenario and system level. By this, we have found out highly sensitive components and connectors that need careful analysis, design, implementation and extra testing effort.

Our future work will focus on developing a performance based risk assessment methodology and generalization of our approach. Another one is to analyze the risk at requirement phase.

Bibliography

[1] Naresh chauhan. *Software testing principles and practices*. Oxford university press India, 2011.

[2] Cem Kaner. Exploratory testing. In *Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL*, 2006.

[3] Hooman Hoodat and Hassan Rashidi. Classification and analysis of risks in software engineering. *World Academy of Science, Engineering and Technology*, 56:446–452, 2009.

[4] Barry W Boehm. Software risk management: principles and practices. *Software, IEEE*, 8(1):32–41, 1991.

[5] C.Ravindranath Pandian. *Applied Software Risk Management a Guide for Software Project Managers*. Auerbach Publications, 2007.

[6] Marvin J Carr, Suresh L Konda, Ira Monarch, F Carol Ulrich, and Clay F Walker. Taxonomy-based risk identification. Technical report, DTIC Document, 1993.

[7] Hu Yong, Chen Juhua, Rong Zhenbang, Mei Liu, and Xie Kang. A neural networks approach for software risk analysis. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 722–725. IEEE, 2006.

[8] John D McGregor and David A Sykes. *Practical guide to testing object-oriented software*. Addison-Wesley Professional, 2001.

[9] Roger S Pressman and Darrel Ince. *Software engineering: a practitioner's approach*, volume 5. McGraw-hill New York, 1992.

[10] Artur Rot. It risk assessment: quantitative and qualitative approach. *Resource*, 283:284, 2008.

[11] Thomas J. McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976.

[12] Sallie Henry and Dennis Kafura. Software structure metrics based on information flow. *Software Engineering, IEEE Transactions on*, (5):510–518, 1981.

[13] Andrew Gemino and Drew Parker. Use case diagrams in support of use case modeling: Deriving understanding from the picture. *Journal of Database Management (JDM)*, 20(1):1–24, 2009.

[14] Vahid Garousi, Lionel C Briand, and Yvan Labiche. Control flow analysis of uml 2.0 sequence diagrams. In *Model Driven Architecture–Foundations and Applications*, pages 160–174. Springer, 2005.

[15] Steven S. Muchnick. *Advanced compiler design implementation*. Morgan Kaufmann Publishers, 1997.

[16] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[17] Martin Hitz and Behzad Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proceedings of the International Symposium on Applied Corporate Computing*, volume 50, pages 75–76, 1995.

[18] Martin Hitz and Behzad Montazeri. Measuring product attributes of object-oriented systems. In *Software EngineeringESEC'95*, pages 124–136. Springer, 1995.

[19] John C Munson and Taghi M Khoshgoftaar. Software metrics for reliability assessment. In *Handbook of Software Reliability Engineering*, pages 493–529. McGraw-Hill, Inc., 1996.

[20] Lionel C. Briand, Sandro Morasca, and Victor R. Basili. Defining and validating measures for object-based high-level design. *Software Engineering, IEEE Transactions on*, 25(5):722–743, 1999.

[21] Sherif M Yacoub, Hany H Ammar, and Tom Robinson. Dynamic metrics for object oriented designs. In *Software Metrics Symposium, 1999. Proceedings. Sixth International*, pages 50–61. IEEE, 1999.

[22] Jesús M Almendros-Jiménez and Luis Iribarne. Describing use-case relationships with sequence diagrams. *The Computer Journal*, 50(1):116–128, 2007.

[23] Jeffrey D Gordon. A uml-based metric approach to estimate total effort using software project designs, 2008.

[24] Ståle Amland. Risk-based testing:: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software*, 53(3):287–295, 2000.

[25] Martin S Feather, Steven L Cornford, Julia Dunphy, and Ken Hicks. A quantitative risk model for early lifecycle decision making. In *6th World Conference on Integrated Design & Process Technology*, 2002.

[26] Steven L Cornford, Martin S Feather, and Kenneth A Hicks. Ddp-a tool for life-cycle risk management. In *Aerospace Conference, 2001, IEEE Proceedings.*, volume 1, pages 1–441. IEEE, 2001.

[27] Sherif M. Yacoub and Hany H. Ammar. A methodology for architecture-level reliability risk analysis. *Software Engineering, IEEE Transactions on*, 28(6):529–547, 2002.

[28] Katerina Goseva-Popstojanova, Ahmed Hassan, Ajith Guedem, Walid Abdelmoez, Diaa Eldin M. Nassar, Hany Ammar, and Ali Mili. Architectural-

level risk analysis using uml. *IEEE Transactions on Software Engineering*, 29(10):946–960, 2003.

[29] K Appukkutty, Hany H Ammar, and Katerina Goseva Popstajanova. Software requirement risk assessment using uml. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 112. IEEE, 2005.

[30] SR Luke. Failure mode, effects and criticality analysis (fmeca) for software. In *5th Fleet Maintenance Symposium*, pages 731–735, 1995.

[31] Vittorio Cortellessa, Katerina Goseva-Popstojanova, Kalaivani Appukkutty, Ajith R Guedem, Ahmed Hassan, Rania Elnaggar, Walid Abdelmoez, and Hany H Ammar. Model-based performance risk analysis. *Software Engineering, IEEE Transactions on*, 31(1):3–20, 2005.

[32] Roger C. Cheung. A user-oriented software reliability model. *Software Engineering, IEEE Transactions on*, (2):118–125, 1980.

[33] C Sundararajan. *Guide to reliability engineering: Data, analysis, applications, implementation, and management*. Van Nostrand Reinhold, 1991.

[34] Sherif M Yacoub, Hany H Ammar, and Tom Robinson. A matrix-based approach to measure coupling in object-oriented designs. *Journal of Object Oriented Programming*, 13(7):8–19, 2000.

[35] Genaína N Rodrigues, David S Rosenblum, and Sebastian Uchitel. Reliability prediction in model-driven development. In *Model Driven Engineering Languages and Systems*, pages 339–354. Springer, 2005.

[36] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 13(1):37–85, 2004.

[37] N.M.Inc. Magicdraw uml v11.6. Website, 2010. `http://www.magicdraw.com`.

[38] Cesar Ortega-Sanchez, Doug Myers, and M Sadi. A novel approach to minimizing the risks of soft errors in mobile and ubiquitous systems. 2009.